ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
DOI: 10.17559/TV-20131113112940

LVM IN THE LINUX ENVIRONMENT: PERFORMANCE EXAMINATION

Borislav Djordjevic, Valentina Timcenko

Original scientific paper
This paper presents a performance evaluation of LVM (*Logical Volume Manager*) system under the Linux operating system. The presented work includes proposal of the mathematical modeling of file system access time with and without LVM option. A mathematical model is further validated for the case of 32bit Linux ext3 file system with kernel version 2.6, taking into consideration the comparison of a file system in two defined configurations, with and without LVM. We have created two LVM options, with the same capacity, but different in complexity of the internal structure. The performance is measured using the Postmark benchmarking software application that simulates workload of Internet mail server. We have defined three types of workloads, generally dominated by relatively small objects. Test results have shown that the best option is to use direct file system realization without applying LVM. Benchmark results are interpreted based on provided mathematical model of file system access time.

Keywords: *disk model; ext3/ext2; file systems; hard disks; journaling; Linux; LVM-Logical Volume Manager*

LVM u Linux okruženju: ispitivanje svojstava

Izvorni znanstveni članak
Ovaj članak predstavlja ocjenjivanje svojstava LVM (*Logical Volume Manager*) sustava pod Linux operativnim sustavom. Rad uključuje matematički model vremena pristupa datotečnom sustavu sa i bez LVM opcije. Matematički model je validiran na primjeru Linux 32bit ext3, na kernel verziji 2.6, a na bazi usporedbe svojstava datotečnog sustava konfiguriranog za dva slučaja, sa i bez LVM. Mi smo generirali dvije LVM mogućnosti, istog kapaciteta, ali različite po složenosti unutarnje strukture. Svojstva su mjerena pomoću Postmark benchmark aplikacije koja simulira opterećenje Internet e-mail servera. Definirali smo tri vrste opterećenja, pri čemu uglavnom dominiraju relativno mali objekti. Rezultati ispitivanja su pokazali da je najbolja opcija da se koristi datotečni sustav bez primjene LVM. Benchmark rezultati su tumačeni na temelju matematičkog modela vrijemena pristupa datotečkom sustavu.

Ključne riječi: *datotečni sustavi; ext3/ext2; journaling; Linux; LVM-Logical Volume Manager; model diska; tvrdi diskovi*

1 Introduction

Linux is a modern, widespread time-sharing operating system that supports a large number of 32-bit and 64-bit journaling-based file systems such as ext3/ext4, ReiserFS, xfs, jfs, btrfs and zfs [1÷5]. Linux supports a VFS (*Virtual File System*) feature, which represents an object-oriented form of file system implementation. It allows to the user the identical access to all files, regardless of file system that these files belong to. LVM (*Logical Volume Manager*) is a novel quality enhancement feature in Linux. LVM partitions have several advantages over standard hard disk partitions. LVM partitions are formatted similar to physical partitions, where one or more physical disks are combined in a form of a disk group. Disk group implies that the total space for storing data is split across one or more logical disks. Logical disks function similarly to standard partitions. They contain the same types of file systems, such as ext3 and appropriate installation point.

For a better understanding of LVM, the physical disk can be imagined as a large number of disk blocks. Several of these disks can be combined in order to make a larger number of blocks, thus making a structure called volume disk group. The volume group can then be split into several smaller logical disks of random size, whereas the new disks or partitions can also be added to logical disks. An administrator can increase or decrease the size of logical disks without destroying the data, which is not the case with standard disk partitions. If the physical disks are in the group of disks on different RAID (*Redundant Array of Independent Disks*) structures, the administrator can widen the logical disk across all the disks in RAID array.

The objective of the paper is to test the influence of LVM on the file system performances. Apart from all the beneficiary characteristics of LVM, such as flexible file system resizing and snapshot options, the predominant benefit is the feature of file system resizing, which means that logical disks can be increased or decreased without the risk of destroying the data. However, LVM brings overheads due to the intensive remapping operations between logical and physical block addresses (additionally burdening the processor operation), thus decreasing the disk performances.

2 Related work and our solution

The implementation and performance research in the field of LVM is one of the most addressed topics in the area of exploring operation system architecture and performances. Thus the related work can be discussed in three directions.

One group of research papers examines comparison of different file systems in Linux environment, predominantly ext3/ext4, xfs, jfs, btrfs and zfs, whereas these architectures are usually tested for systems relying on RAID or non-RAID options [6÷14]. These papers deal with examination of file system characteristics, and mostly propose specific performance-based mathematical modeling, define workloads, testing methodologies, and provide results interpretation [10, 11, 13].

Second research topics cover the comparison of LVM and non-LVM partitions, without LVM snapshot option [15]. These studies rely on the application of different testing methodologies, mainly based on tools such as the benchmark Bonnie ++ or powerful Linux command dd (*disk dump*). Based on the use of Bonnie ++ benchmark it was possible to detected a low decrease in performance of LVM-configured file system when compared to the native file system. Alternatively, the application of the dd

command in LVM file system environment has shown an appreciable (up to 33 %) slowdown in LVM performances. Unfortunately, majority of these papers do not consider interpretation of obtained performance differences, neither provide mathematical modeling of the used system.

Third group of studies provides results related to the performance examination of various LVM snapshot options, mostly considering mutable and immutable snapshots, time chained snapshots, and a range of branching capabilities [16÷18]. Actually, LVM snapshots provide support for backup and recovery mechanisms. Snapshots are based on copy-on-write technology and have remarkable influence on the obtainable write performances. Most of these studies do cover successfully the snapshots behavior analysis, but rarely provide the examination of the difference between LVM without snapshots and non-LVM options.

In this paper we consider the comparison of LVM and non-LVM options, and have chosen ext3 file system as testing environment. First, we have proposed adequate mathematical model for file system for both analyzed architectures, with LVM and without LVM. Then, we have defined proper workloads and performed a set of testing procedures. Finally, based on the obtained results we have provided the analysis of performance differences. The testing procedure relies on the application of the Postmark benchmark, and three different types of test loads. First test focuses on obtaining details related to the performances of system working with small files. The second test explores the performances of the workload with ultra small objects. This way we have put the system into highly loaded and CPU difficult circumstances of performing high number of testing transactions over a large number of extremely small files. Finally we have defined third testing procedure which relies on the workload with increased size of used files (compared to Test 1 workload).

LVM can be created using several different volumes, on the same disks or on different disks. In this paper, the focus was on the case of LVM with one or several volumes, generated on the same physical disks, whereas there is a native ext3 volume, LVM-1, which is the simplest LVM option created from one volume, and LVM-2, representing LVM option created from two volumes. Special attention was given to testing procedures for different LVM levels under fair-play conditions, which refers to testing under the same or similar conditions, in the same hardware environment and under the same operating system. We have considered the same size of the file system for all tested LVM levels and equal amount of free space (approx. 20 GB) for testing, thus we have enabled equal testing conditions.

The particular stand out and contribution of this paper relies on the fact that all the performed tests are based on the specifically developed workloads, testing methodologies and procedures, while the results are further analyzed taking into consideration the contributed LVM/non-LVM mathematical model.

3 LVM technology

A LVM logical space is created in three basic steps [19÷27]. The first step is the selection of the physical memory resources that will be available to LVM for use. Typically, these are standard partitions, and in LVM terminology these physical memory resources are called physical space. The first step in LVM configuration includes the proper initialization of these partitions so that they can be recognized by the LVM system. This includes the proper adjusting of the type of partition (if a physical partition is added) and then an application of a special *pvcreate* command.

Generation of one or more physical spaces initialized for the purpose of LVM represents the second step of logical LVM space creation. It implies the formation of a volume group using the command *vgcreate*. A volume group can be a group of several storages that is composed of one or more physical spaces. When a LVM is active, a new physical space can be added to the volume group.

LVM is further used for the creation of one or more logical spaces by using the created volume group [25, 26]. A classical file system can be created and used for storing data in the created LVM space. For creating the logical space, the *lvcreate* command is used, which also indicates the name and size of the logical space, as well as the name of the volume group that this logical space will be a part of.

3.1 Workload specifications

File-based workloads are designed for testing procedures of file systems, and usually consist of large number of file operations (creation, reading, writing, appending, and file deletion). It comprises of large number of files and data transactions. Workload can be generated synthetically, as a result of applying benchmark software, or as a result of working with some real data applications [28].

Workload characterization is a hard problem, as arbitrarily complex patterns can frequently occur. In particular, some authors chose to emphasize support for spatial locality in the form of runs of requests to contiguous data, and temporal locality in the form of bursty arrival patterns. Some authors [29] distinguish three different arrival processes:

- **Constant.** The interarrival time between requests is fixed;
- **Poisson.** The interarrival time between requests is independent and exponentially distributed;
- **Bursty.** Some of the requests arrive sufficiently close to each other so that their interarrival time is less than the service time.

Instead of mathematical simulated workload, we have applied synthetically generated workload in Postmark benchmark environment.

3.2 Mathematical model for Non-LVM file access time

In this subchapter we present the access time estimation for non-LVM file system in general (regardless of chosen file system) [28÷30]. Expected access time for

non-LVM file system, for specified workload, comprises following components:

$$T_{FSA} = T_{DO} + T_{MO} + T_{FL} + T_{DFA} + T_J . \quad (1)$$

Where T_{FSA} (*File System Access time*) represents total time that workload needs to carry out all operations; T_{DO} (*Directory Operation time*) represents the total time for all operations related to working with directories (search, new object creation and delete of existing objects); T_{MO} (*Metadata Operation time*) represents total time needed for performing metadata operations (metadata search operations, metadata cache forming, and metadata objects modifications); T_{FL} (*Free List time*) presents total time needed for performing operations with free blocks and i-node lists (file expansion/shrinking, creation of new objects or deletion of existing objects); T_{DFA} (*Direct File Access time*) is the time required for direct file blocks operations (read/write); T_J (*Journaling time*) is total time needed for performance of journaling operations, covering metadata writes to the log, and metadata log discharge.

Directory operations T_{DO}, metadata operations T_{MO} and direct file accesses T_{DFA} are cache based accesses and their performances directly depend on cache. Under these conditions, the expected cache service time would be:

$$T_{CST} \approx P_H \cdot \frac{S_R}{R_{CT}} + P_M \cdot T_{DST} . \quad (2)$$

Where Cache Service Time is represented by T_{CST}, hit probability is denoted by P_H, and represents the number of hits in cache versus the total number of requests, while P_M, miss probability, represents the probability of cache miss, counted as number of misses in a cache versus the total number of requests. The cache transfer rate is marked by R_{CT}, S_R is expected request size, while disk service time T_{DST}, is the total time needed to perform disk data transfer.

In the case of *cache-miss*, performances are strictly dependent on disk characteristics T_{DST}, and consist of number of time based components [30]:

$$T_{DST} = T_A + T_M + T_I . \quad (3)$$

Where access time, T_A is total access time needed for mechanical components of disk transfer, T_M is total time needed for write/read operations from disk medium, and interface time T_I is total time needed for read/write operations from disk cache buffer. The access time is further defined as:

$$T_A = T_{CO} + T_{Seek} + T_{Setle} + T_{RL} . \quad (4)$$

Where command overhead time, T_{CO}, is time required for disk commands decoding, seek time T_{Seek} is the time needed for disk servo system positioning, settle time T_{Setle} stands for time required for disk head stabilization, and T_{RL} rotational latency represents time wasted on disk rotation latency.

Actually, there are three dominant components, whose sum can be presented as disk service time T_{DST}, which is the service time for a request related to the disk mechanism.

These components are: (1) the seek time T_{Seek}, which is the amount of time needed to move the disk heads to the desired cylinder; (2) the rotational latency time T_{RL}, which stands for the time that the platter needs to rotate to the desired sector; and (3) the transfer time T_M, is the time to transfer the data from the disk medium to the next higher level. Since these are typically independent variables, we can approximate the expected value of the disk mechanism service time to:

$$T_{DST} = T_{Seek} + T_{RL} + T_M . \quad (5)$$

The transfer time T_M, is a function of two parameters, the transfer rate R_T, which is the transfer rate of data off/onto the disk and S_R which represents the request size. Function can be approximated to:

$$T_M = \frac{S_R}{R_T} . \quad (6)$$

Some variations will occur as a result of track and cylinder switches operations and existence of different track sizes in different disk zones. The rest of this section describes our technique for approximating seek time and rotational latency time parameters.

Seek time. The seek time T_{Seek}, can be approximated as the following function of *dis*, which is the distance calculated based on the number of cylinders to be travelled [30]:

$$T_{Seek}[dis] = \begin{cases} 0 & dis = 0 \\ a + b\sqrt{dis} & 0 \prec dis \le e \\ c + d \cdot dis & dis \succ e \end{cases}, \quad (7)$$

where a, b, c, d and e are device-specific parameters. The parameters a, b, c, d are obtained with an interpolation of the seek function which is obtained after multiple intensive measurements of time needed for positioning of a specific hard drive.

Rotational latency time. If we assume that the requests are randomly distributed on the sectors of the given cylinder using a uniform distribution, than the rotational latency time would be calculated as 1/2 of full rotation time, R_{rotate} according to:

$$T_{RL} = \frac{R_{rotate}}{2} . \quad (8)$$

3.3 Mathematical model for LVM file access time

LVM access time. In the case of LVM based file systems, access time analysis is more sophisticated and given in next subsection. We will start with explanation of LVM anatomy and LVM terminology.

This diagram (Fig. 1) gives an overview of the main elements and architecture of an LVM system [19÷27].

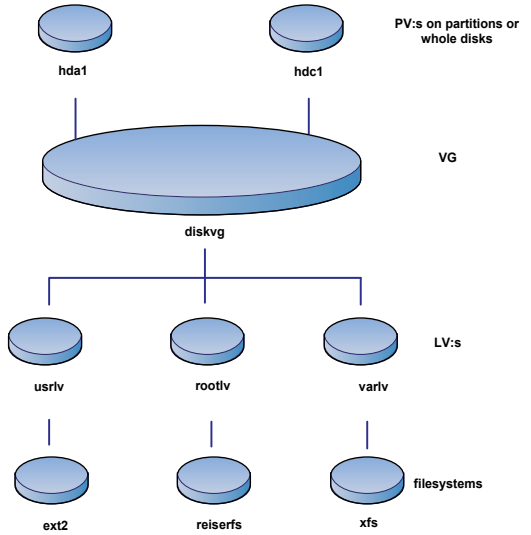

Figure 1 An overview of the main elements in LVM

LVM system encompasses three architectural levels: (1) PV (*Physical Volume*) which includes physical disks or physical disk partitions (e.g. /dev/sda, /dev/sdb1); (2) VG (*Volume Group*) level which combines available physical volumes PVs in one single unit capable of adding new PVs or removing existing PVs; (3) LV (*Logical Volume*) level allows the user creation of logical partitions with common file systems as ext3, ext4, xfs, or jfs. Thus, LVM practically allows higher flexibility, scalability and online resizing.

Volume Group is the highest level abstraction used within the LVM. It gathers together a collection of Logical Volumes and Physical Volumes into one administrative unit.

Logical Volume is the equivalent of a disk partition in a non-LVM system. The LV is visible as a standard block device; as such the LV can contain a file system.

Physical Extent each PV is divided chunks of data, known as physical extents (PE), and these extents have the same size as the logical extents for the VG.

Logical Extent each logical volume is split into chunks of data, known as logical extents (LE). The extent size is the same for all logical volumes in the VG.

LE to PE mapping. The administrator can choose between several general strategies for mapping logical extents onto physical extents, like following 2:

- ***Linear mapping*** will assign a range of PE's to an area of an LV in order
- ***Striped mapping*** will interleave the chunks of the logical extents across a number of physical volumes.

LVM mapping between logical and physical extents (*LE to PE mapping*) is the basic factor that will have an impact to the LVM and Non-LVM file system performance differences. The Device Mapper Driver has a functionality of providing LVM mapping of the logical volumes to the disk physical blocks. For each logical volume the DMD keeps record of the device mapping table based on which it maps the request to specific physical disk block.

DMD provides different forms of mapping: Linear, Striped, Snapshot, and Snapshot-Origin. Each mapping form implies the redirection of the IP requests to a specific underlying physical device. Additionally, the DMD module maintains the information on mapping from the logical volume to the underlying physical volumes by generating and maintaining mapping table. The mappings will differ based on the chosen type of LVM.

When booting, LVM will first scan the metadata stored within the physical volumes of the defined volume group, with an aim to proceed with the DMD registration of each logical volume and its corresponding mapping table. It will end with its registration within the adequate kernel. Thus, DMD redirects any input/output from the logical volume to the defined physical block device.

The LVM access time $T_{\text{LVM\_AT}}$ is calculated according to the following equation:

$$T_{\text{LVM\_AT}} = T_{\text{FSA}} + T_{\text{LVM\_MO}}. \tag{10}$$

LVM mapping overhead time $T_{\text{LVM\_MO}}$ is function of the following components: number of physical volumes, physical PVs disk partitioning approach, size and physical positions of PVs, and volume group partitioning to logical volumes:

$$T_{\text{LVM\_MO}} = f(N_{\text{D}}, N_{\text{FS}}, S_{\text{FS}}, L_{\text{FS}}), \tag{11}$$

Where N_{D} represents the number of disks, N_{FS} is number of file systems, S_{FS} is the size of file system, and L_{FS} represents the location of file system. LE to PE mapping is performed for all file system objects: files, directories, as well as for metadata objects (i-node tables, external attributes (i-node indirect block), bit maps or free block linked lists). Based on that concept, LVM mapping overhead can be presented through the sum of two main components: metadata object mapping overhead MD_{mapping}, and file blocks mapping overhead FB_{mapping}.

$$T_{\text{LVM\_MO}} = f(MD_{\text{mapping}}, FB_{\text{mapping}}). \tag{12}$$

Actually, MD_{mapping} is metadata mapping factor, and FB_{mapping} represents file block mapping factor. It is to be expected as the result of testing procedures that LVM mapping overhead will bring some decrease in file system performances.

The Eqs. (1) to (12) are applicable to any journaling/LVM Linux based file systems (ext3/ext4/jfs/xfs/btrfs), but it is necessary to take into consideration that these file systems differ in achievable quality of time metrics defined in equations.

3.4 Hypotheses on the expected LVM performances

The preliminary hypotheses based on the proposed mathematical model are:

H1: It is expected that the non-LVM over performs LVM
H2: H1 relies in LVM mapping overhead, Eq. (10)
H3: Mapping overhead depends on the complexity of the LVM implementation (number/type of physical volumes)
H4: Mapping overhead depends on the nature of the workloads (read/write/random/sequential components).

It is assumed that the mapping overhead depends on the applied file system (ext3, ext4, xfs, jfs, btrfs), but it is out of scope of this paper and will be left for future work.

Although on the basis of Eq. (10), we expect that LVM slows the performances, we must be careful. In the area of the computer techniques many things are not obvious as they can look like. For example, let's take the journaling technique into consideration. Many potential users will think that journaling techniques with their additional log writes would slow down the performances, but experience shows that this is not always the case. Many papers that can be found as open literature, including some of our own researches [9], show that journaling can also accelerate the performance. Thus, from these experimental researches we could learn that nothing can be taken as obvious, but it has to be proven.

A very similar situation can occur in the case of LVM/Non-LVM environments. We have tried to very precisely examine: (1) whether LVM slows down or even improves the performances; (2) what are the circumstances when it slows down; and (3) what are the parameters that this trend depends on.

4 Mathematical model validation and result analysis

The validation of the proposed mathematical model, as well as the confirmation of stated hypotheses is achieved through a synthetic benchmarking for 32bit ext3, which is one of the most stable, fast and widely accepted 32bit journaling file systems, with a long history of bug-fixes and performance improvements. We emphasize that this file system is still relevant, as there is a large number of ext3 file system users. Just to mention that ext3 file system is often present in Linux Xen virtualization, which shows superior results in comparison to competitors (KVM, VMware, Hyper-V). However, such a Linux Xen-based virtualization is typical for some older Linux distributions, where ext3 was the default file system (FS). Although the newer ext4 file system would be an interesting topic for LVM examination, there is also a long list of possible FS candidates for creating LVM volumes: 32bit ext2/ext3, and different 64bit file systems (ext4, xfs, jfs, btrfs). As we have thoroughly examined the LVM vs. non-LVM for ext3, we believe that it would be interesting for further work to proceed with same testing procedure for the mentioned 64bit candidates, and we expect that the results would confirm our hypotheses, especially H1 and H3 (that LVM introduces performance decrease when compared to native file system, and that these decelerations are greatly affected by the complexity of the LVM configuration).

4.1 Test configuration

For testing purposes, we have chosen disks from the WDC WD800BD-22MR-80GB series (80 GB, average seek time 8,9 ms, rotational speed 7200 rpm, maximum disk buffer throughput 1,5 Gb/sec) and Red Hat Linux version Fedora 10 with kernel version 2.6.27. For the purpose of testing, the file system is organized in the form of logical partitions, where the last part of disk (20GB) is created in three different ways and used for testing.

In the first case, direct file system realization was created without the use of LVM (/dev/sda3 of 20GB in ext3 format), as it is provided in Tab. 1.

Table 1 FS layout without LVM

File system	Size	description
LogVol00	58GB	root FS
LogVol01	1GB	Swap
dev/sda3	20GB	testing FS

In the second case, LVM is made of one physical volume (/dev/sda3 as a physical volume of 20 GB and logical group LogVol02 of 20GB in ext3 format), as provided in Tab. 2.

Table 2 FS layout with an LVM composed of one PV, LVM-1

File system	Size	Description
LogVol00	58GB	root FS
LogVol01	1GB	Swap
dev/sda3	20GB	physical volume
LogVol02	20GB	testing FS

In the third case, LVM is composed of two physical volumes (/dev/sda3 and /dev/sda4 as physical volumes of 10GB and logical group LogVol02 of 20 GB in ext3 format). Details are given in Tab. 3.

For second and third case, an empty ext3 file system is created in a logical group LogVol02. LogVol02 is specially created for the purpose of testing, and it can be accessed through the path /dev/mapper/VolGroup00-LogVol02. The file system used for testing is of exactly the same size for all the tests and tested LVM levels.

Table 3 FS layout with an LVM composed of two PVs, LVM-2

File system	Size	Description
LogVol00	58 GB	Root FS
LogVol01	1 GB	Swap
dev/sda3	10 GB	physical volume
dev/sda4	10 GB	physical volume
LogVol02	20 GB	testing FS

4.2 Testing procedures and analysis

For the purpose of this paper we have used PostMark [31], software that simulates loading an Internet Mail server. PostMark creates a large initial pool of randomly generated files in any place in the file system. This pool is further used for creating, reading, writing and file deleting, and the time required for these operations is determined. The sequence of these operations is random therefore the simulation credibility is guaranteed. The number of files, their size range and the number of transactions are fully configurable. In order to eliminate the cache effect it is recommendable to create an initial pool with as many files as possible (at least 10 000) and perform as many transactions as possible.

We have presented the results of three different test procedures. Test 1 is based on testing of small files (file sizes between 1KB and 100KB) and this test results will be a reference when comparing other test results. Test 2 considers drastically smaller files (sizes between 1byte and 1KB) and appreciably increased number of generated files, which will generate high number of metadata operations with ultra-small objects. Test 3 considers slightly increased size of generated files when comparing to Test 1, which implies higher dataflow in workload.

4.2.1 Postmark Test 1

Files used for the purpose of performing this testing procedure are relatively small, ranging from 1KB to 100 KB. The results are given in Tab. 4 and on Fig. 2. In this test of small files, direct file system realization without the use of LVM shows superior performances in comparison to the two tested LVM configurations. Direct file system realization without the use of LVM is about 63 % faster than an LVM-1 realization with one physical volume, and it is 2,5 times faster than an LVM-2 realization with two physical volumes. LVM-1 realization with one physical volume is approximately 50 % faster than an LVM-2 realization with two physical volumes.

The workload for this test is characterized with lower number of files (4000), moderate number of create/delete operations, file size of 1 k ÷ 100 k, and solid amount of reads/writes (1.6GB-r/1.8GB-w). Having into consideration the equation (1), it can be expected that component T_{DFA} will be dominant.

Table 4 Results for Postmark Test 1

MB/s	ext3	LVM-1	LVM-2
Read	2,99	1,83	1,21
Write	3,49	2,14	1,42

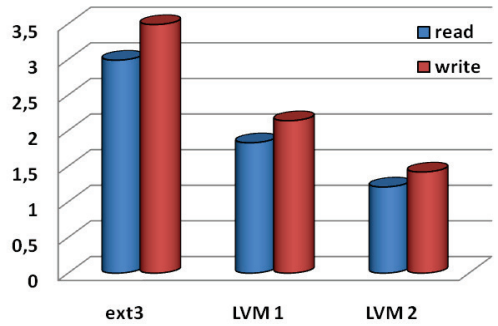

Figure 2 PostMark Test 1 results

Starting from Eq. (11), we will consider that this test assumes both kinds of mapping (metadata object mapping overhead and file blocks mapping overhead), and that both of them have an impact to the decrease of LVM performances. In Test 1, block remapping overhead has a dominant influence on the decrease of LVM options performance when comparing to native ext3.

4.2.2 Postmark Test 2

This is also a very intensive test procedure as it involves a large number of very small files, ranging from 1bytes to 1KB. The test generates a large number of metadata I/O requests. The results are given in Tab. 5 and on Fig. 3. In this test of ultra-small files, direct file system realization without the use of LVM shows superior performance in comparison to the two tested LVM configurations. Direct file system realization without the use of LVM is approximately 80 % faster than an LVM-1 realization with one physical volume and it is 6,5 times faster than a LVM-2 realization with two physical volumes. LVM-1 realization with one physical volume is significantly faster (3,5 times) than an LVM realization with two physical volumes.

Table 5 Results for Postmark Test 2

KB/s	ext3	LVM-1	LVM-2
read	149,82	81,96	23,03
write	345,15	188,82	53,06

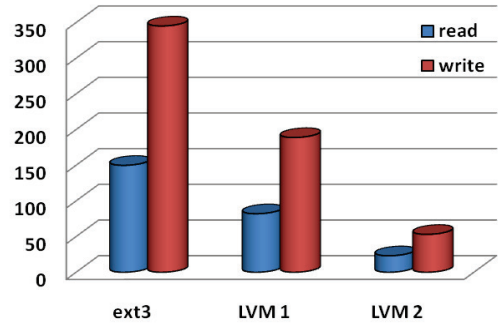

Figure 3 PostMark Test 2 results

The workload for this test is characterized with high number of files (30 000), high number of create/delete operations, ultra-small file sizes (1byte-1K), and low amount of reads/writes (13.61MB-r/31.35MB-w). Having into consideration Eq. (1), it can be expected that components sum $T_{DO} + T_{MO}$ will be dominant. Taking into consideration Eq. (11), in this specific case, as there is large number of metadata operations (noticeable larger than in Test 1), metadata object mapping overhead component has dominant influence to the remarkable LVM performance decrease. In the case of LVM-2 it is evident that the amount of metadata mappings was significantly greater than in the case LVM-1, resulting in significant decrease of LVM-2 performances.

4.2.3 Postmark Test 3

This is a very intensive test. Files used for this testing procedure are relatively large (1KB-300KB). Results are given in Tab. 6 and on Fig. 4.

In the test of larger files, direct file system realization without the use of an LVM also shows superior performance when comparing to LVM configurations.

Table 6 Results for Postmark Test 3

KB/s	ext3	LVM-1	LVM-2
read	2,8	2,4	0,91
write	3,25	2,79	1,03

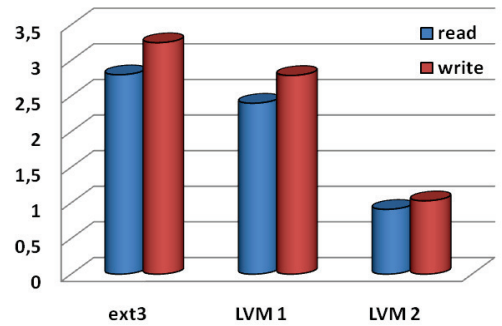

Figure 4 PostMark Test 3 results

Native file system realization without the use of LVM is approximately 16 % faster than an LVM realization with one physical volume, and it is 3 times faster than an LVM realization with 2 physical volumes.

LVM realization with one physical volume is significantly faster (2,5 times) than an LVM realization with 2 physical volumes.

The workload for this test is characterized with lower number of files (4000), moderate number of create/delete operations, file size of 100 k ÷ 300 k, and solid amount of reads/writes (4.6GB-r/5.4GB-w). Having into consideration the Eq. (1), it can be expected that T_{DFA} component will be dominant. Again, it is worth considering Eq. (11), and notice that in Test 3, as the size of tested files has increased in comparison to the files generated in Test 1, there is appreciably higher number of file blocks, while the number of metadata operations is approximately the same as in Test 1. Therefore, the file blocks mapping overhead in LVM has dominant role to the very significant decreasing of LVM performances. The more complex file block remapping procedure for increased number of file blocks in the case of LVM-2 has caused appreciable decrease in performance when compared to LVM-1, and this difference is more intense than the one detected in Test 1.

4.3 General remarks for all three Postmark tests

In this paper, we have presented the results of PostMark tests for (1) basic file system without the use of LVM, and (2) two LVM realizations with one or two physical volumes. Our goal was to consider different working circumstances, by first examining a workload of small files, and then in the second test substantially increased the number of metadata operations, while in the third experiment we have considerably increased the number of blocks for file transfer.

Taking into consideration all the obtained results, as the confirmation of the preliminary hypotheses H1 ÷ H4, it can be observed that the native non-LVM realization is solidly better than LVM, while LVM with one or two physical volumes shows significantly lower performance in 2 out of 3 tests: in the test with ultra-small files and in the test with somewhat larger files, in each case with huge number of disk metadata or direct file access operations.

5 Conclusion

The obtained experimental results fully confirm the set of preliminary hypotheses and validate the proposed mathematical model. The beneficial characteristics of LVM should be highly appreciated, as LVM provides the flexible widening of file systems, with new partitions on the same or different disks, during which there is no data loss. But it has to be taken care of the fact that LVM represents a new layer in the virtual file system hierarchy, which creates overheads for processing, thus has a negative impact on performances. The compensation of the LVM mapping overhead is partially performed by file-caching features, although it is always present in some form. This is confirmed by the provided test results. In general we can conclude that:

- LVM provides reduced level of performances when compared to non-LVM file system, which confirms H1.
- Decrease in performances highly depends on the complexity of LVM configuration, thus confirming H3. LVM shows greater loading and weaker performance if realized in a more complex way, meaning that if an LVM is created from several different volumes it is possible to achieve significantly lower performances. It is also expected that the performances will further decrease if a LVM is created from volumes located on different physical disks, which was not directly tested in this paper and will be discussed in some further work.
- The performance decrease depends directly on the workload, number of the required disk operations, metadata operations and number of direct file block operations (confirmation of the H4 hypothesis).
- The main cause of the noticed performance decline is the feature of remapping. The remapping in the case of the LVM is performed by the DMD. This procedure is weakly compensated by the remapping caching. The obtained results confirm the H2 hypothesis.

It is pointed out to the users that the complex LVM implementation should be an interim solution, while final one should be the LVM with the simplest possible design or non-LVM at all.

Our LVM test methodology fits well when dealing with QoS (*Quality of Service*) issues related to virtualization and cloud computing, as the concept of LVM is often seen as file system options for Linux based host and guest operating systems. Therefore, LVM could have a large performance impact on virtual environments, different cloud computing environments, as well as to data centers. Many Linux users or administrator will linearly proceed with expand their storage space of existing workstations/servers by simply adding new hard drives, thus obtaining the needed expansion in rapid and non-destructive way. Their existing LVM volumes will be expanded with new disks and partitions without data loss, which is a relatively quick procedure and is in compliance with the primary purpose of LVM (easy storage expansion/shrinking). But, in that way there is a risk of creating relatively complex LVM designs. In this paper, it is proved that the complex LVM can suffer from significant performance decline, and showed that this quick and easy solution based on the LVM volume spreading should be just an interim solution, not the final one. The suggested solution is based on first proceeding with long-term full backup operation, following with reconfiguration of the used LVM to the simplest possible design. Finally, after these steps it is recommendable to run a full restore procedure. Alternatively, in the presence of the performance-critical applications, it is recommendable to proceed with the complete replacement of the LVM with available native file system.

Future work will encompass performance analysis of more sophisticated LVM types, with focus on LVM case when implementing multiple partitions and multiple physical disks. It will cover LVM performances on the 64bit file systems as an enhancement to the results presented in this paper, as well as LVM on the RAID, LVM with activated snapshots options and LVM in the virtual and cloud computing environment.

Acknowledgements

The presented work has been funded by the Serbian Ministry of Education, Science and Technological Development (TR32037, TR43002 and TR 32025).

6 References

[1] Seltzer, M.; Ganger, G.; McKusick, M.; Smith, K.; Soules C.; Stein C. Journaling versus Soft Updates: Asynchronous Meta-data Protection in File Systems. // Proceedings of the USENIX Conference / San Diego, 2000, pp. 71-84.
[2] Silberschatz, A.; Galvin, P. Operating System Concepts. Addison-Wesley, 2007.
[3] Avantika, M..; Cao, M.; Bhattacharya, S.; Dilger, A.; Tomas, A.; Vivier, L. et al. The new ext4 filesystem: current status and future plans. // Proceedings of the Linux Symposium, vol. 2 / Ottawa, Canada, 2007, pp. 21-34.
[4] Tweedie, S. EXT3, Journaling Filesystem. 2000, 7.
[5] Hagen, B. Exploring the ext3 Filesystem. Linux Planet, 2002, 4. URL: http://www.linuxplanet.com/linuxplanet/
[6] reports/4136/1 (5.04.2002).
[7] Machado, A. LVM, RAID, XFS and EXT3 file systems tuning for small files massive heavy load concurrent parallel I/O on Debian. // Quarta-Feira, 4 de Julho de 2012.
[8] ZFS vs. EXT4 on Linux Multi-Disk RAID Benchmarks. Phoronix, 25 April 2013, The Web version 4(2013)
[9] Phromchana V.; Napuairoj N.; Piromsopa K. Performance Evaluation of ZFS and LVM (with ext4) for Scalable Storage System. // 8th Int. Joint Conf. On Computer Science and Software Engineering (JCSSE) / Mahidol University, NakhonPathom, Thailand, 2011, pp. 250-253. DOI: 10.1109/jcsse.2011.5930130
[10] Djordjevic, B.; Timcenko, V. Ext4 file system in Linux Environment: Features and Performance Analysis. // International Journal of Computers. 6, 1(2012), pp. 37-45.
[11] Piernas, J.; Cortes, T.; García, J. M. Dual FS: a new journaling file system without meta-data duplication. // Proc. of int. conference on Supercomputing (ICS '02). ACM, New York, 2002, pp. 137-146. DOI: 10.1145/514191.514213
[12] Lee, E.; Yoo, S.; Jang, J-E.; Bahn, H. Shortcut-JFS: A write efficient journaling file system for phase change memory. // Symposion on Mass Storage Systems (MSST) 2012, pp. 1-6.
[13] Vickovic, L.; Mudnic, E.; Gotovac, S. Parity information placement in the disk array model. // COMPEL - The Int. J. for Computation and Mathematics in Electrical and Electronic Engineering. 28, 6 (2009), pp. 1428-1441. DOI: 10.1108/03321640910991994
[14] Sehgal, P.; Tarasov, V.; Zadok, E. Evaluating performance and energy in file system server workloads. // Proc. 8th USENIX conference on File and storage technologies (FAST'10). / USENIX Association, Berkeley, CA, USA, 2010, pp. 19-19.
[15] Dagenais, M. R. Disks, partitions, volumes and RAID performance with the Linux operating system. // Computing Research Repository, 2005.
[16] Smorul, M. LVM vs normal disk partitioning speed testing with Bonnie ++. URL: http://www.umiacs.umd.edu/~toaster/lvm-testing/
[17] Nayak, P.; Ricci, R. Detailed study on Linux Logical Volume Manager. // Flux Research Group University of Utah, August 1, 2013.
[18] Shah, B. Disk Performance of Copy-On-Write Snapshot Logical Volumes (Master's Thesis). // The University of British Columbia, 2006.
[19] Radhakrishnan, P. Stateful-Swapping in Emulab network testbed (Master's Thesis). // The University of Utah, 2008.
[20] The original version of LVM. LVM Resource Page. URL : https://www.sourceware.org/lvm
[21] Danen, V. Set up Logical Volume Manager in Linux. // Techrepublic online, Mar 09 2007. URL: http://www.techrepublic.com/article/set-up-logical-volume-manager-in-linux/
[22] Toft, G. Using Logical Volume Management. // Linux Journal. URL: http://www.linuxjournal.com/article/5957. (23.09.2002.)
[23] Scalzo, B. Optimizing Oracle 10g on Linux: Non-RAC ASM vs. LVM. // Linux Journal (31.08.2005.). URL: http://www.linuxjournal.com/article/8539
[24] Bulling, R. Recovery of RAID and LVM2 Volumes. // Linux Journal URL: http://www.linuxjournal.com/article/8874 (28.04.2006).
[25] Lewis, A. J. LVM HOWTO // The Linux Documentation Project, 2006.
[26] Pillai, S. The Advanced Guide to LVM-Logical Volume Management on Linux: PART1. // Slashroot.in Magazine, 2013
[27] Red Hat, Red Hat Ent. Linux6, Logical Volume Manager Administration LVM Administrator Guide, Ed.1, 2013.
[28] Red Hat Linux: What is Logical Volume Manager or LVM? // Technology Magazine (6.08.2013.).
[29] Smith, K. A. Workload-Specific File System Benchmarks. PhD thesis. // Harvard University Cambridge, Massachusetts, 2001.
[30] Shriver, E.; Merchanty, A.; Wilkesy, J. An analytic behaviour model for disk drives with readahead caches and request reordering. // Proceedings of the ACM SIGMETRICS joint Int. conference on Measurement and modelling of computer systems / 26, 1(1998), pp. 182-191.
[31] Akpinar, O. Performance Analysis of the Disk Subsystem. Rutgers // The State University of New Jersey, New Brunswick, 2006.
[32] Katcher, J. PostMark: A New File System Benchmark. // Technical Report TR3022 / Network Appliance Inc (1997).

Authors' addresses

Borislav Djordjevic, PhD, Associate Professor
University of Belgrade,
Institute Mihailo Pupin, Computer Systems department
Volgina 15, 11060 Belgrade, Serbia
E-mail: bora@impcomputers.com

Valentina Timcenko, Magister, Associate researcher
University of Belgrade,
Institute Mihailo Pupin, Telecommunications department
Volgina 15, 11060 Belgrade, Serbia
E-mail: valentina.timcenko@pupin.rs